Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks

William Robertson, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer
Reliable Software Group
Department of Computer Science
University of California, Santa Barbara
{wkr,vigna,chris,kemm}@cs.ucsb.edu

Abstract

The custom, ad hoc nature of web applications makes learning-based anomaly detection systems a suitable approach to provide early warning about the exploitation of novel vulnerabilities. However, anomaly-based systems are known for producing a large number of false positives and for providing poor or non-existent information about the type of attack that is associated with an anomaly.

This paper presents a novel approach to anomaly-based detection of web-based attacks. The approach uses an anomaly generalization technique that automatically translates suspicious web requests into anomaly signatures. These signatures are then used to group recurrent or similar anomalous requests so that an administrator can easily deal with a large number of similar alerts.

In addition, the approach uses a heuristics-based technique to infer the type of attacks that generated the anomalies. This enables the prioritization of the attacks and provides better information to the administrator. Our approach has been implemented and evaluated experimentally on real-world data gathered from web servers at two universities.

1. Introduction

In the past ten years, the World-Wide Web has evolved from a system to provide access to static information into a full-fledged distributed execution infrastructure. Web-based applications have become a popular way to provide access to services and dynamically-generated information. The popularity of web-based applications, such as online shopping catalogs and web-based discussion forums, is a result of the ease of development, deployment, and access of this class of applications. Even network devices and traditional applications (such as mail servers) often provide web-based interfaces that are used for administration as well as configuration.

Unfortunately, while the developers of the software infrastructure (that is, the developers of web servers and database engines) usually have a deep understanding of the security issues associated with the development of critical software, the developers of web-based applications often have little or no security skills. These developers mostly focus on the functionality for the end-user and often work under stringent time constraints, without the resources (or the knowledge) necessary to perform a thorough security analysis of the application code. The result is that poorly-developed code, riddled with security flaws, is deployed and made accessible to the whole Internet.

Web-related security flaws represent a substantial portion of the total number of vulnerabilities. This claim is supported by an analysis of the vulnerabilities that have been made public in the past few years. For example, by analyzing the Common Vulnerabilities and Exposures (CVE) entries from 1999 to 2005 [5], we identified that web-based vulnerabilities account for more than 25% of the total number of security flaws. Note that this is only a partial account of the actual number of web-based vulnerabilities, since there are a number of

ad hoc web-based applications that have been developed internally by companies to provide customized services, and many of the security flaws in these applications have not yet been discovered or made public.

Because of their immediate accessibility, poor security, and large installation base, web-based applications have become popular attack targets and one of the main venues to compromise the security of systems and networks. Preventing attacks against web-based applications is not always possible, and, even when suitable mechanisms are provided, developers with little security training (or simply with little time) sometimes disable security mechanisms "to get the job done." Therefore, prevention mechanisms should be complemented by effective intrusion detection systems (IDSs).

To detect web-based attacks, intrusion detection systems are configured with a number of "signatures" that support the detection of known attacks. These systems match patterns that are associated with the exploitation of known web-related flaws against one or more streams of events obtained by monitoring web-based applications [1, 14, 16, 21]. For example, at the time of writing, Snort 2.3.3 [16] devotes 1064 of its 3111 signatures to detecting web-related attacks. Unfortunately, it is hard to keep intrusion detection signature sets updated with respect to the new vulnerabilities that are continuously being discovered. In addition, vulnerabilities may be introduced by custom web-based applications developed in-house. Developing *ad hoc* signatures to detect attacks against these applications is a time-intensive and error-prone activity that requires substantial security expertise.

Anomaly detection [6, 8, 9, 12] is an approach to intrusion detection that is complementary to the use of signatures. Anomaly detection relies on models of the normal behavior of users and applications to identify anomalous activity that may be associated with intrusions. The main advantage of anomaly-based techniques is that they are able to identify previously unknown attacks. By defining the expected, normal behavior, any abnormality can be detected, whether it is part of a known attack or not.

Anomaly detection can be performed by applying different techniques to characterize the normal behavior of a target system. Of particular interest for the detection of attacks against web-based applications are learning-based techniques, which build a model of the normal behavior of an application by observing the usage patterns of the application during a training period. Once the model of normal behavior is established, the IDS switches to "detection mode" and compares the behavior of the application with respect to the model learned during the training period, with the assumption being that anomalous behavior is likely to be associated with an intrusion (and that an intrusion will result in anomalous behavior).

Learning-based techniques are particularly suitable for the detection of web attacks, because they can detect attacks against custom-developed code for which there are no known signatures or attack models. These systems can also operate in unsupervised mode, with little or no input from system administrators and application developers. Therefore, they can be used by administrators that have little security training.

For example, consider a custom-developed web-based application called `purchase`, where the identifier of the item to be purchased (`itemid` parameter) and the credit card type (`cc` parameter) are inserted by the user in a client-side form and then validated by a server-side application invoked through the Common Gateway Interface [4]. A set of sample invocations of the `purchase` application, as logged by the web server, is shown in Figure 1. In this case, a learning-based anomaly detection system can build a model of the `itemid` and `cc` parameters that are passed to the application by analyzing a number of normal purchase transactions. The models could characterize the length of the parameters, their character distribution, or their structure. Once these models have been learned by analyzing a number of samples, each subsequent invocation of the `purchase` application is compared to the established models, and anomalies in the parameters are identified. For example, the last entry in Figure 1 would be identified as an attack, because the `itemid` parameter has a structure (and length) that is anomalous with respect to the established models.

Even though anomaly-based detection systems have the potential to provide effective protection, there are two main problems that need to be addressed. First, learning-based anomaly detection systems are prone to producing a large number of false positives. Second, anomaly detection systems, unlike misuse-based systems, only report that there is an anomaly without any supporting description of the attack that has been de-

```
128.111.41.15 "GET /cgi-bin/purchase?itemid=1a6f62e612&cc=mastercard" 200
128.111.43.24 "GET /cgi-bin/purchase?itemid=61d2b836c0&cc=visa" 200
128.111.48.69 "GET /cgi-bin/purchase?itemid=a625f27110&cc=mastercard" 200
131.175.5.35  "GET /cgi-bin/purchase?itemid=7e2877b177&cc=amex" 200
161.10.27.112 "GET /cgi-bin/purchase?itemid=80d2988812&cc=visa" 200
...
128.111.11.45 "GET /cgi-bin/purchase?itemid=109agfe111;ypcat%20passwd|mail%20wily@evil.com" 200
```

Figure 1. Sample log entries associated with invocations of the `purchase` application. The last entry represents an attack against the application.

tected.

The goal of the work reported in this paper is to overcome both of these problems. We have developed a system that is based on an "anomaly generalization" mechanism that derives a generalized representation of the anomalies detected by a learning-based intrusion detection system. The result is an "anomaly signature" that is used to identify further occurrences of similar anomalies. Similar anomalies are grouped together and are then analyzed by the administrator to determine if each group, as a whole, is composed of false positives or actual attacks. If the alerts in a group are identified as being false positives, then they can be dismissed with a single decision, saving a considerable amount of the administrator's time. Also, the anomaly signature characterizing the group can be used as a suppression filter to prevent further false positives or as a new training data set to improve the anomaly detection models. If the alerts in a group are identified as instances of an actual attack, then these alerts can be used as the basis to either identify and fix a security flaw or to develop a "traditional" attack signature.

To address the problem of poor attack explanatory information, our system uses a heuristics-based technique to infer the type of attack that generated the anomaly. Our previous experience with the detection of web-based attacks showed that while custom-developed server-side code might be exploited using unpredicted attribute values, the *type* of exploitation often follows specific rules. Therefore, we developed heuristics that can identify common classes of attacks, such as buffer overflows, cross-site scripting, SQL injection, and directory traversals. Note that this characterization is different from a misuse detection signature, because our heuristics are applied only to the portion of an event that caused the anomaly (e.g., the value of a specific parameter). Therefore, if there are other parts of the data that could appear as an attack but that are actually benign (that is, they are normal according to the established profile), then our characterization will not generate a false positive, while a misuse detection signature probably would.

In summary, our anomaly detection system can be deployed on an existing web-based system, and in an unsupervised fashion can characterize the normal behavior of server-side components. It can then detect deviations from the established profile, group similar anomalies, and, in some cases, give an explanation of the type of attack detected.

This paper is structured as follows. Section 2 discusses related work and the limitations of current intrusion detection systems in their ability to detect web-based attacks. Section 3 presents the architecture for our system and briefly describes its main components. Section 4 presents the anomaly models used by the anomaly detector. Section 5 describes our approach to anomaly generalization. Section 6 discusses the characterization of certain types of anomalies and the heuristics that we use. Section 7 provides an evaluation of the approach in terms of the overhead introduced, the reduction in the number of false positives, and the ability to appropriately characterize attacks. Finally, Section 8 draws conclusions and outlines future work.

2. Related Work

The work presented here is related to three different areas of intrusion detection: learning-based anomaly detection, application-level intrusion detection, and the detection of attacks against web servers. In the following,

we discuss how previous work in these three areas relate to our research.

Different types of learning-based anomaly detection techniques have been proposed to analyze different data streams. A common approach is to use data-mining techniques to characterize network traffic. For example, in [15], the authors apply clustering techniques to unlabeled network traces to identify intrusion patterns. Statistical techniques have also been used to characterize user behavior. For example, the seminal work by Denning [6] builds user profiles using login times and the actions that users perform.

A particular class of learning-based anomaly detection approaches focuses on the characteristics of specific applications and the protocols they use. For example, in [7] and in [3], sequence analysis is applied to system calls produced by specific applications in order to identify "normal" system call sequences for a certain application. These application-specific profiles are then used to identify attacks that produce previously unseen sequences. As another example, in [13], the authors use statistical analysis of network traffic to learn the normal behavior of network-based applications. This is done by analyzing both packet header information (e.g., source/destination ports, packet size) and the contents of application-specific protocols.

Our approach is similar to these techniques because it characterizes the benign, normal use of specific programs (i.e., server-side web-based applications). However, our approach is different from these techniques in two ways. First, we use a number of different models to characterize the parameters used in the invocation of the server-side programs. By using multiple models it is possible to reduce the vulnerability of the detection process with respect to mimicry attacks [18, 23]. Second, the models target specific types of applications, and, therefore, they allow for more focused analysis of the data transferred between the client (the attacker) and the server-side program (the victim). This is an advantage of application-specific intrusion detection in general [10] and of web-based intrusion detection in particular [11].

The detection of web-based attacks has recently received considerable attention because of the increasingly critical role that web-based applications are playing. For example, in [1] the authors present a system that analyzes web logs looking for patterns of known attacks. A different type of analysis is performed in [2] where the detection process is integrated with the web server application itself. In [21], a misuse-based system that operates on multiple event streams (i.e., network traffic, system call logs, and web server logs) was proposed. Systems that focus on web-based attacks have demonstrated that by taking advantage of the specificity of a particular application domain it is possible to achieve better detection results. However, these systems are mostly misuse-based and therefore suffer from the problem of not being able to detect attacks that have not been previously modeled.

In [19], the authors propose a serial architecture where web-related events are first passed through an anomaly detection component. Then, the events that are identified as neither normal nor intrusive are passed on to a misuse detection component. The system proposed in [19] is different from our approach because it requires extensive manual analysis to evaluate the characteristics of the events being analyzed. Our goal is to require the minimum amount of manual inspection and human intervention possible. Therefore, we focus on techniques to perform unsupervised, learning-based anomaly detection.

3. Architecture

The goal of detecting unknown attacks against custom-developed software as well as characterizing and grouping these attacks in a meaningful way necessitated the development of a novel approach to intrusion detection. In this approach, an event collector and anomaly detection component are composed to provide the ability to detect unknown attacks. In addition, three new components, an *anomaly aggregation* component, an *anomaly signature generation* component, and an *attack class inference* component, are introduced into the architecture. The integration of these components into the design allows the resulting system to harness the strengths of anomaly detection while mitigating its negative aspects. An overview of the architecture is depicted in Figure 2.

The event collector first creates and normalizes the events. The normalized events are then passed to the anomaly detector, which determines whether the event is anomalous or not. If an event is normal, no alert is generated. If the event is anomalous, on the other hand,

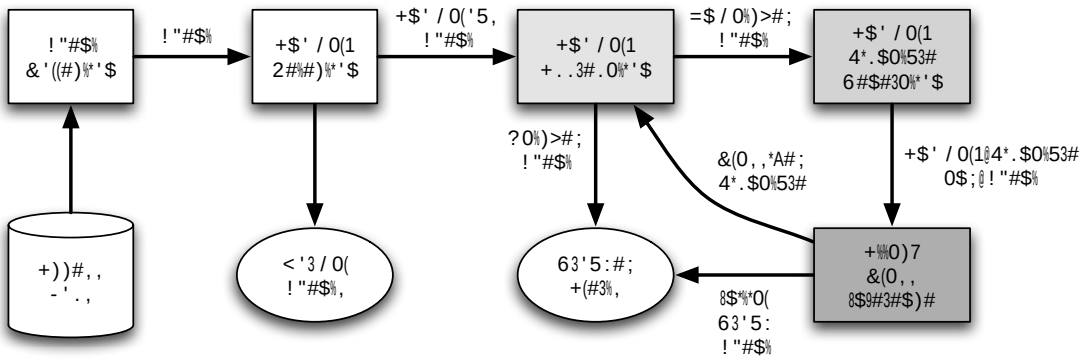

Figure 2. Architecture of web intrusion detection system.

it is passed to the anomaly aggregation component. This component matches the event against a set of "anomaly signatures." The idea is to determine whether an event is similar to a previously detected anomaly. If an event can be matched with an anomaly signature, an alert is generated immediately and grouped with instances of previous similar alerts. Otherwise, the event is passed to the anomaly signature generation process. The task of this component is to generalize the anomaly and to construct an appropriate "anomaly signature." Finally, the attack class inference component attempts to determine the class of the anomaly, using a set of heuristics. The resulting, classified anomaly signature is added to the set of existing "anomaly signatures," and an alert is generated for the anomalous event. The individual components of the architecture are discussed in more detail in the following sections.

3.1. Event Collection

The event collection component is responsible for capturing and preparing individual events from the operating environment, for analysis by the intrusion detection components. Events may be collected from several sources (e.g., a network link, a system call auditing facility embedded into a web server's host operating system, or the access logs generated by a web server). For the work presented in this paper, the events were collected from web server access logs, but the approach itself is agnostic as to the source of the events.

3.2. Anomaly Detection

The anomaly detection component examines the subset of web server access log entries that represent parameterized web requests. These requests specify a web application and a set of attribute names and corresponding values to be passed as parameters to the application. During an initial learning phase, the anomaly detector constructs a characterization of normal invocations of server-side applications by analyzing the observed attribute values. This process is performed using several different models, each of which focuses on a specific feature of an attribute (e.g., its length or its character distribution). These models are used to construct profiles specific to each attribute for each server-side resource. For example, an attribute length model could be used to build a profile that specifies the normal length of the value for the attribute `itemid` when used in the invocation of the `purchase` web application in Figure 1. The details of this process are discussed in Section 4.

Once a profile for an attribute of a web application has been constructed, the anomaly detection component switches from the learning phase to the detection phase for that attribute. In this phase, the anomaly detector evaluates the observed values of the attribute for a given request with respect to the established profile. This process outputs an anomaly score associated with the request, and if the anomaly score exceeds a model-specific detection threshold, an alert is generated. The assumption is that highly anomalous queries (e.g., a request

where the value of the `itemid` attribute of the aforementioned `purchase` application is unusually long) could represent evidence of an attempt to exploit a vulnerability of a web-based application. By utilizing an anomaly detection component that automatically builds profiles for web applications in this manner, it is possible to detect previously unknown attacks without requiring the development of *ad hoc* signatures.

3.3. Anomaly Aggregation

The anomaly aggregation component processes the stream of anomalous events from the anomaly detector, matching them against a set of anomaly signatures maintained within the component. This signature set is modified at run-time; in particular, new signatures produced by the anomaly signature generation and attack class inference components can be dynamically integrated into the existing set in the midst of event processing. If an event (or series of events, in the case of a stateful, multi-step anomaly signature) matches an anomaly signature in this set, an alert is generated and grouped with previous instances of similar alerts.

3.4. Anomaly Signature Generation

Anomalous events that are not matched by any anomaly signature in the anomaly aggregation component are forwarded to the anomaly signature generation component. The task of this component is to construct a generalized *anomaly signature* that is based on the particular anomaly and the web application and attribute that was the target of the request. The generated signature is used to match further manifestations of the same, or similar, anomaly. Note that a distinction must be made between *misuse signatures* and *anomaly signatures*. In our nomenclature, a misuse signature describes a specific instance of a known attack, whereas an anomaly signature is a set of statistical models of suspected malicious behavior for a specific target that has been derived from an alert generated by an anomaly detector.

3.5. Attack Class Inference

In certain cases, a signature can be further classified as belonging to one of several broad, generic classes of known attacks against web-based applications. In our system, these classes currently include buffer overflows, cross-site scripting, SQL injection, and directory traversal. The task of the attack class inference component is to use a set of heuristics to assign an attack class to an anomaly signature.

The classification of attacks can result in semantically rich anomaly alerts that are more informative to security administrators, as well as to web application developers. These alerts not only pinpoint the vector through which the application is being attacked (i.e., the specific attribute), but the nature of the attack as well. This can assist the security administrator or application developer in quickly mitigating the vulnerability, as the nature of the attack itself generally suggests the required course of action.

The following sections examine the details of how each of these components are implemented.

4. Anomaly Detection

The anomaly detection component utilizes a number of statistical models to identify anomalous events of a set of web requests that use parameters to pass values to an associated web application. The original prototype and the models used to characterize the requests were introduced in [11] and are summarized here to give to the reader the background necessary to understand the generalization techniques presented in Section 5.

The anomaly detection component operates on the URLs extracted from successful web requests, as provided by the event collector. The set of URLs is further partitioned into subsets corresponding to each web application (that is, each server-side component). The anomaly detector processes each subset of queries independently, associating models with each of the attributes used to pass input values to a specific server-side application.

The anomaly detection models are a set of procedures used to evaluate a certain feature of a query attribute, and operate in one of two modes, learning or detection. In the learning phase, models build a profile of the "normal" characteristics of a given feature of an attribute (e.g., the normal length of values for an attribute), setting a dynamic detection threshold for the attribute. During the detection phase, models return an anomaly score for each observed example of an attribute value. This is

simply a probability on the interval $[0, 1]$ indicating how probable the observed value is in relation to the established profile for that attribute (note that a score close to zero indicates a highly anomalous value).

Since there are generally multiple models associated with each attribute of a web application, a final anomaly score for an observed attribute value during the detection phase is calculated as the weighted sum of the individual model scores. If the weighted anomaly score is greater than the detection threshold determined during the learning phase for that attribute, the anomaly detector considers the entire request anomalous and raises an alert.

The following sections briefly describe the implementation of the models utilized by the anomaly detector component. For more information on the models themselves as well as the anomaly detector as a whole and its evaluation on real-world data, please refer to [11].

4.1. Attribute Length

The attribute length model is based on the assumption that many attribute values do not vary greatly in length, being either fixed in size or of a short length. Malicious input, however, often violates this assumption, such as in the case of a buffer overflow which must pass a string, often containing shellcode, that is long enough to overflow the target buffer in order to assume control of the vulnerable process. Thus, the attribute length model attempts to approximate the actual, yet unknown, distribution of the attribute lengths. This is accomplished during the learning phase by calculating the sample mean μ and variance σ^2 for each attribute value observed. During the detection phase, the probability that a given attribute length l is drawn from the actual distribution of attribute lengths is calculated using the formulation based on the Chebyshev inequality, where l is the observed attribute length, μ is the sample mean, and σ^2 is the sample variance.

$$p\left(|x-\mu| > |l-\mu|\right) < p\left(l\right) = \frac{\sigma^2}{\left(l-\mu\right)^2}$$

The weak bound enforced by the Chebyshev inequality results in a high degree of tolerance to variations in attribute length, flagging only obvious outliers as suspicious.

4.2. Character Distribution

The character distribution model is motivated by the observation that attribute values generally have a regular structure, are usually human-readable, and usually only contain printable characters. In particular, values for a given attribute can be expected to have similar character distributions, where a character distribution is composed of the relative frequencies of each of the 256 ASCII character values sorted in descending order. For legitimate inputs, the relative character frequencies are expected to slowly decrease in value. Malicious input, however, can exhibit either an extreme drop-off due to large occurrences of a single character, or little drop-off due to random character values.

During the learning phase, the *idealized character distribution*, or ICD, of an attribute is calculated by first recording the character distribution for each observed example. Before switching to the detection phase, the average of all recorded character distributions for that attribute are computed. During the detection phase, the probability that the character distribution of an observed attribute value is drawn from the calculated ICD for that attribute is determined using a variant of the Pearson χ^2-test.

4.3. Structural Inference

The structural inference model derives its power from the observation that legitimate attribute values can often be considered as strings generated from a regular grammar. For instance, a subset of legal pathnames could be generated from the regular expression `(/|[a-zA-Z0-9])+`, i.e., a series of alphanumeric characters interspersed with path separators. The generating grammar for an attribute, however, is unknown, and thus it is necessary to construct a reasonable approximation for the true grammar. This is accomplished during the learning phase by considering the observed attribute values as the output of a *probabilistic grammar*, which is a grammar that assigns probabilities to each of its productions. The probabilistic grammar captures the notion that some strings are more likely to be produced than others, and should correspond to the set of examples gathered during the learning phase.

The construction of such a grammar is accomplished by the application of the algorithm described in [17].

This algorithm first constructs a nondeterministic finite automaton (NFA) that exactly reflects the input data, and then gradually merges states until a reasonable generalization from the starting grammar is found, at which point state merging terminates. The goal is to find a middle ground between the over-simplified starting grammar, which is only able to derive the learned input, and an over-generalized grammar which is capable of producing all possible strings (in which case all structural information has been lost). The resulting NFA associates probability values with each of the symbols emitted and the transitions taken between states, with the probability of a single path through the automaton being the product of those probabilities.

During the detection phase, the probability that an observed attribute value has been generated from the true generating grammar for that attribute is calculated as the product of the probabilities for the symbols emitted and transitions taken along a path through the NFA. If no path is possible, the observed value cannot be derived from the probabilistic grammar, and a probability of 0 is returned.

4.4. Token Finder

The token finder model depends on the observation that some attributes expect values drawn from a limited set of constants, such as flags or indices. Thus, this model detects when an attacker attempts to use these attributes to pass values not contained in the legal set to the application. During the learning phase, the set of unique values for a given attribute are recorded. If the size of this set grows proportionally to the total number of observed instances of the attribute, the expected values for the attribute are assumed to be random. Otherwise, the model assumes that the attribute expects an enumeration of values. During the detection phase, if the attribute has been determined to accept an enumeration of values, observed attribute values are tested for membership in the set recorded during the learning phase. If the observed value is present, then the model assumes that the value is innocuous. Otherwise, the observed value is considered anomalous.

5. Anomaly Generalization

Anomaly generalization is the process of transforming the parameters of a detected anomaly into an abstract model that will be used to match similar anomalies, where the similarity metric is model-dependent.

More precisely, when one or more of the attribute values of a web request are detected as anomalous by one or more models, the detection parameters for the attributes and models involved are "relaxed" and composed in an anomaly signature that identifies possible variations of an attack. For example, if the attribute `itemid` of the `purchase` web application shown in Figure 1 is detected as anomalous because of its character distribution, then the generalization process will create an abstract model that matches character distributions for the `itemid` attribute that are somewhat similar to the one that triggered the anomaly.

Note that the generalization process and the generation of anomaly signatures is not driven by examples of known attacks, but by the relaxation of the parameters used by the anomaly models. Therefore, these anomaly signatures are not derived from (or associated with) a specific exploitation technique.

The following sections describe the details of how our system generalizes anomalies detected by each of the models discussed in Section 4, and how these generalized anomaly signatures are used during the anomaly aggregation phase.

5.1. Attribute Length

The attribute length model stores the approximate length distribution of an attribute derived during the learning phase as a sample mean μ and variance σ^2. When the length of a given attribute value is detected as anomalous, the values of μ and σ^2 for that attribute are extracted from the model. These parameters are then used to create an anomaly signature, which determines whether lengths for the same attribute value are similarly anomalous.

To this end, we introduce a similarity operator $\psi_{attrlen}\left(l_{obsv}, l_{orig}\right)$ where

$$\psi_{attrlen} \equiv \left| \frac{\sigma^2}{\left(l_{obsv} - \mu\right)^2} - \frac{\sigma^2}{\left(l_{orig} - \mu\right)^2} \right| < d_{attr}$$

This operator is adapted from the Chebyshev inequality test and is used during the anomaly aggregation phase to determine whether the anomaly score of an observed attribute length l_{obsv} falls within some configurable distance d_{attr} from the anomaly score of the anomalous attribute length l_{orig} that was originally used to generate the anomaly signature.

5.2. Character Distribution

The character distribution model stores the idealized character distribution (ICD) of an application's attribute, and then, during the detection phase, it applies the Pearson χ^2-test to determine the normality of an attribute's character distribution. If the distribution is flagged as anomalous, the parameters that are necessary to create the generalized anomaly signature are extracted in one of two ways, depending on the nature of the anomaly.

1. If the observed character distribution exhibits a sharp drop-off, which indicates the dominance of a small number of characters, a configurable number of the character values that dominate the distribution are used to construct the generalized signature. More formally, the set $C = \{(c_1, f_1), (c_2, f_2), \ldots, (c_m, f_m)\}$ is constructed, where c_i is the i^{th} dominating character value, f_i is the corresponding relative frequency, and C is the set of m dominating character values and frequency pairs extracted from the model. During the anomaly aggregation phase phase, the similarity of an observed character distribution with respect to the original anomalous character distribution is tested by analyzing the intersection between C_{obsv} and C_{orig}, where C_{obsv} is the set of dominating characters from the observed attribute value and C_{orig} is the corresponding set from the original anomalous value. If $C_{obsv} \cap C_{orig} \neq \emptyset$, we introduce a similarity operator $\psi_{cdist}(f_{obsv,i}, f_{orig,j})$ where

$$\psi_{cdist} \equiv |f_{obsv,i} - f_{orig,j}| < d_{cdist}$$

and

$$(c_{obsv,i}, f_{obsv,i}), (c_{orig,j}, f_{orig,j}) \in C_{obsv} \cap C_{orig},$$

$$c_{obsv,i} = c_{orig,j}$$

Here, d_{cdist} is a configurable distance threshold. Thus, two dominating character sets are considered similar if at least one character value is present in their intersection and the corresponding relative frequencies are within a configurable distance from each other.

2. If the character distribution of the anomalous attribute is close to the uniform distribution, the similarity operator becomes a test for a nearly random character distribution. This is implemented by calculating the maximum distance between any pair of frequency values from the sets C_{obsv} and C_{orig}, and by testing whether this distance is less than a configurable threshold d. Formally, $\forall\, 0 \leq i, j \leq m$

$$\psi_{cdist} \equiv max\,(|f_{obsv,i} - f_{orig,j}|) < d_{cdist}$$

Note that these two different techniques are necessary to accommodate common attacks (such as injection of binary code and directory traversal attacks), which manifest themselves as a character distribution anomaly but with very different characteristics.

5.3. Structural Inference

The structural inference model uses the examples observed in the training phase to construct a probabilistic grammar that approximates the actual grammar for the values of an application's attribute.

If an attribute value observed during the detection phase is determined to be anomalous by the model, the generalization process extracts the prefix of the violating string up to and including the first character that violates the attribute's grammar. The use of the offending string's prefix as a base for generalizing the attack is motivated by the observation that repeated attacks against the same web application often exhibit similar prefixes in the URLs or paths used as values in that attribute of the application.

The prefix string is translated into a string of character classes. More precisely, all lowercase alphabetic characters are mapped into "`a`," all uppercase alphabetic characters are mapped into "`A`," all numeric characters map to "`0`," and all other characters remain unchanged. Then, to test the similarity of a subsequent observed value with respect to the translated anomalous string,

called s_{orig}, a similar translation is performed on the observed value, which becomes s_{obsv}. The two normalized values are then compared for equality. Formally, we introduce the similarity operator $\psi_{structure}\,(s_{obsv}, s_{orig})$ such that $\forall\, 0 \leq i \leq m$ where $m = |s_{orig}|$,

$$\psi_{structure}\,(s_{obsv}, s_{orig}) \equiv s_{obsv,i} = s_{orig,i}$$

For example, consider the value of the attribute `itemid` in the last line of Figure 1. In this case, the structural inference model detects a violation in the structure of the attribute value because, during the training phase, the model learned that an item identifier is composed of alphanumeric characters only. Therefore, the generalization process creates an anomaly signature based on the grammar `[a|0]+;`, because the character "`;`" is the first character that violated the attribute grammar derived during the training phase. Further attempts to use the "`;`" character to perform an attack will be classified as similar anomalies.

5.4. Token Finder

During the detection phase, the token finder tests an observed attribute value l_{orig} for membership in the set T, where T is the set of tokens recorded during the learning phase for that attribute (if the set of all values for that attribute was determined to be an enumeration).

During the detection phase, if the token finder model determines that an attribute value is anomalous, the set of allowable values T for that attribute is extracted from the model. Then, during the aggregation phase, a subsequent observed attribute value l_{obsv} is detected as similar to the original anomalous value l_{orig}, if the observed value is not a member of the enumeration T and it is lexicographically similar to the original anomalous value.

Formally, given a function lex that determines lexicographic similarity, we introduce the similarity operator $\psi_{token}\,(l_{obsv})$ where

$$\psi_{token} \equiv lex(l_{orig}, l_{obsv})$$

Note that the function lex can be tuned to achieve different levels of sensitivity to variations in the values of anomalous tokens. For instance, lex may use Hamming or Levenshtein distances to determine string equality. Additionally, type inference may be performed on the collection and an appropriate lex function selected. In the degenerate case, lex always returns true and the resulting anomaly signature would simply group anomalies that represent attribute values not contained in the original enumeration. In the current implementation, however, lex is a simple string equality test.

For example, in the web application shown in Figure 1, the `cc` attribute always has values drawn from the set of credit card types {`mastercard`, `visa`, `amex`}. If an exploit uses an anomalous value for this attribute, the corresponding anomaly signature will identify identical violations of the attribute value.

6. Attack Class Inference

Anomaly detection is able to detect unknown attacks but it is not able to provide a concrete explanation of what the attack represents with respect to the target application. This is a general limitation of anomaly detection approaches and often confuses system administrators when they have to analyze alerts that state only that some attribute of a web request did not match one or more of the previously established profiles. We observed that certain well-known classes of attacks violate anomaly models in a consistent way. Therefore, whenever such violations are detected, heuristics can be used to attempt to infer the type of an attack and provide useful hints to the system administrator.

Our system includes an attack class inference component that utilizes *ad hoc* heuristics to determine the class of an attack when certain types of anomalies are detected. The selection of which heuristics to apply as well as how each is applied is influenced by the type and parameters of the anomaly detected. Our system currently incorporates heuristics for four major classes of web-based attacks: directory traversals, cross-site scripting, SQL injection, and buffer overflows.

As noted in Section 1, the attack class inference process is different from the matching of "traditional" intrusion detection signatures (e.g., a Snort signature that detects buffer overflow attacks). The reason is that the attack class inference process is applied only to attributes that have already been identified as anomalous, while "traditional" signatures are applied to the entire event being analyzed. As a consequence, the attack class inference technique can be more abstract (and less precise) without incurring the risk of classifying benign portions of an event as malicious. Note that the attack class in-

ference is performed in addition to (and independent of) the derivation of a generalized anomaly signature, and does not always produce a valid classification.

In the following, we describe how attack classes are identified for the four classes of attacks currently supported.

6.1. Directory Traversal

Directory traversal attacks are essentially attempts to gain unauthorized access to files that are not intended to be accessed by a web application or web server by traversing across the directory tree using `..` and `/` escapes. These attacks are somewhat unique in that a small set of characters is involved in their execution, namely "`.`" and "`/`". Accordingly, the heuristics for detecting directory traversals are only activated if either the character distribution returns a dominating character set C where $C \cap \{., /\} \neq \emptyset$, or if the structural inference model returns a violating character-compressed string with a final underivable character of "`.`" or "`/`." To infer the presence of a directory traversal attack, the heuristic scans the anomalous attribute value for a substring derivable by the regular grammar represented by `(/|\.\.)+`.

For example, suppose that the `purchase` web application of Figure 1 is invoked with an `itemid` value of "`cat ../../../../../etc/shadow`". In this case, the character distribution model identifies an anomalous number of "`.`" and "`/`" characters, and, in addition, the structural model detects a violation of the attribute structure. As a consequence, the directory traversal attack class inference heuristics are applied to the anomalous attribute value. The heuristics determine that the attribute matches the regular expression `(/|\.\.)+`, and the attack is identified as a directory traversal attack.

This information is added to the generalized signature associated with the anomalous event, and this anomalous event, as well as further similar anomalous events, are presented to the system administrator as a group of anomalies labeled as directory traversal attacks.

6.2. Cross-site Scripting

Cross-site scripting attacks allow a malicious user to execute arbitrary code on a client-side machine by injecting malicious code, such as a JavaScript script, into a web document (e.g., storing JavaScript code in a database field) such that the code is unwittingly served to other clients. These attacks generally consist of fragments of client-side browser scripting languages. Because of the insertion of specific HTML tags and the use of code-like content, this type of attack often results in a violation of the structural inference, character distribution, and token finder models.

Consequently, the cross-site scripting heuristics are applied to an anomalous attribute value if any of these models are involved in the initial detection step. The heuristics currently used for this class include a set of scans for common syntactic elements of the JavaScript language or HTML fragments (e.g., `script` or left or right angle brackets used to delimit HTML tags).

6.3. SQL Injection

SQL injection attacks consist of unauthorized modifications to SQL queries, usually by escaping an input to a query parameter that allows the attacker to execute arbitrary SQL commands. Because of the insertion of these escape characters, SQL injection attacks generally result in the violation of the structure of an attribute.

Thus, the heuristics specific to SQL injection are activated if the structural inference model detects an anomaly. The heuristics themselves perform a set of scans over the attribute value for common SQL language keywords and syntactic elements (e.g., `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `'`, or `--`).

6.4. Buffer Overflows

Buffer overflow attacks, which encompass attacks such as stack smashing, heap smashing, data modification, and others, typically involve sending a large amount of data that overflows the allocated buffer, allowing the attacker to overwrite return addresses, data or function pointers, or otherwise overwrite sensitive variables with attacker-controlled data. Buffer overflow attacks against web applications typically manifest themselves as attribute values that deviate dramatically from established profiles of normalcy. Thus, the heuristics for inferring the presence of a buffer overflow attack will be activated if any of the character distribution, structural inference, or attribute length models report an at-

tribute as anomalous. The heuristics in the current system perform a simple scan over the attribute string for binary values (i.e., ASCII values greater than 0x80), which are typical of basic buffer overflow attacks. More sophisticated classification techniques could be substituted or used to supplement the basic heuristic, such as abstract execution [20], with associated tradeoffs in performance.

7. Evaluation

The system was evaluated in terms of its false positive rate, its ability to correctly group and classify anomalies, and its ability to perform detection on web request logs in real-time. All experiments were conducted on a Pentium IV 1.8 GHz machine with 1 GB of RDRAM.

7.1. False Positive Rate

In order to evaluate the false positive rate of the anomaly detector, data sets from two universities, TU Vienna and UCSB, were analyzed by the system. To this end, a client was written to replay the requests to a honeypot web server while a misuse detection system sniffed a link between the client and server. All requests corresponding to reported attacks were stripped from the data set. Also, since many of the attacks were intended for Microsoft IIS while the data sets were produced by the Apache web server, many attacks were stripped out simply by removing requests for documents that did not exist.

The detection system itself was configured with an initially empty anomaly signature set, and default learning, detection, and similarity thresholds were used. The learning phase was performed over the first 1,000 examples of a specific web application attribute, at which point the attached profile was switched into detection mode. During detection mode, any alerts reported by the system were flagged as false positives, due to the assumption that the data set was attack-free. The results of the experiment are shown in Table 1.

During analysis of the TU Vienna data set, the detection system produced 14 alerts over 737,626 queries, resulting in a quite low false positive rate. We believe this attests to the ability of the anomaly detection models to accurately capture the "normal" behavior of attribute values during the learning phase. The addition of the anomaly generalization and aggregation components, however, improved this even further by allowing the system to collapse those 14 alerts into 2 groups. When these groups were examined, we found that each of the groups indeed comprised related alerts. For the first, an IMAP mailbox was repeatedly accessed through the `imp` webmail application, which had not been observed during the learning phase. In response, the token finder generated an alert, and the resulting anomaly signature allowed the system to group the alerts together in a logical manner.[1] For the second group, developers of a custom web application passed invalid values to an attribute during test invocations of their program. In this case, the attribute length model detected an anomaly, and the resulting anomaly signature correctly grouped subsequent variations on the input errors with the first instance.

The results of the generalization and aggregation components during analysis of the UCSB data set were even more dramatic. The detection system reported 513 alerts over 35,261 queries, resulting in a false positive rate several orders of magnitude greater than the TU Vienna data set. However, due to generalization and aggregation, the 513 alerts were partitioned into 3 groups. Manual inspection of the aggregated alerts demonstrated that, as in the case of the TU Vienna data set, the groups were again comprised of related alerts. The first group was composed of a series of anomalous queries to the `whois.pl` user lookup script, which expects a `name` attribute with a valid username as the value. In this case, the grouped alerts all possessed the `name` attribute value `teacher+assistant++advisor`, possibly as the result of a bad hyperlink reference to the script from elsewhere on the department website. In this case, the character distribution model detected an anomalous number of "`a`" characters. The second group was identical in nature to the first group, except that the name argument value was `dean+of+computer+science`. For this group, the character distribution detected an anomalous number of "`e`" characters. The final group was composed of several alerts on an optional argument to the `whois.pl` script named `showphone`, which takes either a `yes` or `no` value as an argument. In this case, the alerts were attributed to an uppercase `YES`,

[1] Incidentally, this would be a reasonable case to put the associated models back into the learning phase, in order to incorporate the characteristics of the legitimate value into the attribute profile.

Table 1. False positive results.

Data set	Queries	False positives	False Positive Rate	Groups	Grouped False Positive Rate
TU Vienna	737,626	14	1.90×10^{-5}	2	3.00×10^{-6}
UCSB	35,261	513	1.45×10^{-2}	3	8.50×10^{-5}

Table 2. Attack classification results.

Attack	Detected?	Variations	Groups	Alerting Models	Characterization
csSearch	Yes	10	1	Length, Char. Distribution	Cross-site scripting
htmlscript	Yes	10	1	Length, Structure	Directory traversal
imp	Yes	10	1	Length, Char. Distribution	Cross-site scripting
phorum	Yes	10	1	Length, Char. Distribution, Token	Buffer overflow
phpnuke	Yes	10	1	Length, Structure	SQL injection
webwho	Yes	10	1	Length	None

which the token finder correctly detected as anomalous.

To evaluate the effectiveness of the attack inference heuristics, a number of attacks comprising the different attack classes that the system claims to detect, were injected into the TU Vienna data set. This data set was chosen because legitimate invocations of the vulnerable applications were previously present in the access log. Ten variations of each distinct attack were injected throughout the data set, utilizing mutation techniques from the Sploit framework [22]. The detection system was configured with exactly the same parameters as in the previous experiment. The results of the experiment are shown in Table 2.

From the experimental results, we first note that all instances of each attack were determined to be anomalous by the anomaly detector. This is to be expected, as the main focus of this work is on the effectiveness of grouping and characterizing related anomalous alerts, and not on improving the ability of the system to detect raw anomalies. In each case, all instances of a given attack were classified into one group. In addition, each of the groups was correctly characterized as belonging to the proper attack class. The only attack that was not characterized by the attack inference heuristics was the `webwho` attack. This, however, is correct behavior from the system, as the `webwho` attack exploited an input validation error for which the system includes no characterization heuristics. It is important to note, however, that the anomaly was still detected, and further variations were grouped correctly. Indeed, although a variety of models provided the initial decision that the request was anomalous, in each case the anomaly signature generation procedure was able to match subsequent variations of the same attack. We believe that this demonstrates the power our anomaly generalization technique, specifically with respect to its ability to group similar anomalies.

7.2. Performance

The performance of the detection system was evaluated in terms of both elapsed processing time and memory usage when run on both attack-free data sets from TU Vienna and UCSB. Both metrics are important for the real-world applicability of this system, since in the ideal case it would be run in real-time on hardware available to most web site operators. The same parameters used for the false positive evaluation were used for this experiment. Ten runs were performed for each data set, and the elapsed times were averaged. The results of the time required for analysis by the system are displayed in Table 3.

For both data sets, the detection system was able to

Table 3. Detection performance results (time).

Data set	Requests	Request Rate	Elapsed Analysis Time	Analysis Rate
TU Vienna	737,626	0.107095 req/sec	934 sec	788.06 req/sec
UCSB	35,261	0.001360 req/sec	64 sec	550.95 req/sec

maintain a processing rate orders of magnitude above the rate of client requests logged by the web server. For instance, in the case of the TU Vienna data set, the request analysis was performed approximately 7,000 times as quickly as actual requests were being logged. From this, we conclude that for many sites, the detection system is capable of performing its analysis in real-time.

In addition to CPU usage, an analysis of the memory utilization of the system was performed. The results of this evaluation showed that the system did not require substantial memory resources once the profiles were established. The details of the memory usage evaluation are not provided for the sake of space.

8. Conclusions and Future Work

This paper presented an approach that addresses the limitations of anomaly-based intrusion detection systems by using both generalization and characterization techniques. Generalization is used to create a more abstract description of an anomaly that enables one to group similar attacks. Characterization is used to infer the class of attack that is associated with a group of anomalies. Using these two techniques, it is possible to reduce the time required by an administrator to make decisions about the nature of the anomalies (actual attacks versus false positives) and their criticality. Furthermore, the generalization and characterization presented can assist application developers in pinpointing the location and nature of previously unknown vulnerabilities.

One possible drawback of the architecture occurs if an attack that has been successfully detected is grouped with attacks that will be considered false positives. If the group of attacks is dropped by the system administrator, then the real attack is not identified as such and becomes a false negative.

We developed a system that implements anomaly signature generation and attack class inference, and we tested it on real-world data collected at two universities. The results shows that the proposed techniques are able to correctly generalize and characterize attacks, considerably reducing the effort needed to analyze the output of the intrusion detection system.

The promising results of our initial experiments suggest that the generalization and characterization techniques can be extended to other domains, such as the arguments of system calls issued by critical applications. Future research will explore these new domains and the general applicability of our techniques.

We will also investigate whether the attack inference technique can be improved, either by using more sophisticated heuristics or by relying on different decision models. For example, we plan to explore whether attack characterization can be expressed as a Bayesian network where the model outputs are used as evidence nodes.

Finally, we also plan to investigate whether evaluation of the system using alternative metrics increases the precision of our characterization of the system's effectiveness in reducing the effective false positive rate.

Acknowledgments

This research was supported by the Army Research Office, under agreement DAAD19-01-1-0484, and by the National Science Foundation, under grants CCR-0238492 and CCR-0524853.

References

[1] M. Almgren, H. Debar, and M. Dacier. A Lightweight Tool for Detecting Web Server Attacks. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, San Diego, CA, February 2000.

[2] M. Almgren and U. Lindqvist. Application-Integrated Data Collection for Security Monitoring. In *Proceed-*

ings of Recent Advances in Intrusion Detection (RAID), LNCS, pages 22–36, Davis, CA, October 2001. Springer.

[3] C. Warrender and S. Forrest and B.A. Pearlmutter. Detecting Intrusions using System Calls: Alternative Data Models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[4] K. Coar and D. Robinson. The WWW Common Gateway Interface, Version 1.1. Internet Draft, June 1999.

[5] Common Vulnerabilities and Exposures. `http://www.cve.mitre.org/`, 2005.

[6] D.E. Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, February 1987.

[7] S. Forrest. A Sense of Self for UNIX Processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, May 1996.

[8] A.K. Ghosh, J. Wanken, and F. Charron. Detecting Anomalous and Unknown Intrusions Against Programs. In *Proceedings of the Annual Computer Security Application Conference (ACSAC'98)*, pages 259–267, Scottsdale, AZ, December 1998.

[9] C. Ko, M. Ruschitzka, and K. Levitt. Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specifi cation-based Approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 175–187, Oakland, CA, May 1997.

[10] C. Kruegel, T. Toth, and E. Kirda. Service Specifi c Anomaly Detection for Network Intrusion Detection. In *Symposium on Applied Computing (SAC)*. ACM Scientifi c Press, March 2002.

[11] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the* 10^{th} *ACM Conference on Computer and Communication Security (CCS '03)*, pages 251–261, Washington, DC, October 2003. ACM Press.

[12] W. Lee, S. Stolfo, and P. Chan. Learning Patterns from Unix Process Execution Traces for Intrusion Detection. In *Proceedings of the AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, July 1997.

[13] M. Mahoney and P. Chan. Learning Nonstationary Models of Normal Network Traffi c for Detecting Novel Attacks. In *Proceedings of the* 8^{th} *International Conference on Knowledge Discovery and Data Mining*, pages 376–385, 2002.

[14] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, January 1998.

[15] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion Detection with Unlabeled Data Using Clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security*, Philadelphia, PA, November 2001.

[16] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the USENIX LISA '99 Conference*, Seattle, WA, November 1999.

[17] A. Stolcke and S. Omohundro. Inducing Probabilistic Grammars by Bayesian Model Merging. In *Conference on Grammatical Inference*, 1994.

[18] K.M.C. Tan, K.S. Killourhy, and R.A. Maxion. Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits. In *Proceedings of the* 5^{th} *International Symposium on Recent Advances in Intrusion Detection*, pages 54–73, Zurich, Switzerland, October 2002.

[19] E. Tombini, H. Debar, L. Me, and M. Ducasse. A Serial Combination of Anomaly and Misuse IDSes Applied to HTTP Traffi c. In *Proceedings of the Twentieth Annual Computer Security Applications Conference*, Tucson, Arizona, December 2004.

[20] Thomas Toth and Christopher Kruegel. Accurate Buffer Overfbw Detection via Abstract Payload Execution. In *5th Symposium on Recent Advances in Intrusion Detection (RAID)*, 2002.

[21] G. Vigna, W. Robertson, V. Kher, and R.A. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, pages 34–43, Las Vegas, NV, December 2003.

[22] Giovanni Vigna, William Robertson, and Davide Balzarotti. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *11th ACM Conference on Computer and Communications Security (CCS)*, 2004.

[23] D. Wagner and P. Soto. Mimicry Attacks on Host-Based Intrusion Detection Systems. In *Proceedings of the* 9^{th} *ACM Conference on Computer and Communications Security*, pages 255–264, Washington DC, USA, November 2002.